PCT/DE 99/00213

BUNDESREPUBLIK DEUTSCHLAND

09/601167

PRIORITY DOCUMENT

SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

EJV #1

Bescheinigung

DE 99/00213

Die Anmelderin Bundesrepublik Deutschland, vertreten durch das Bundesministerium der Verteidigung, vertreten durch Bundesamt für Wehrtechnik und Beschaffung in Koblenz/Deutschland hat eine Patentanmeldung unter der Bezeichnung

"Verfahren und Einrichtung zur komprimierten Übertragung und/oder Speicherung von einer durch digitale Daten repräsentierten Nachricht"

am 31. Januar 1998 beim Deutschen Patent- und Markenamt eingereicht.

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

Die Anmeldung hat im Deutschen Patent- und Markenamt vorläufig das Symbol H 03 M 7/30 der Internationalen Patentklassifikation erhalten.

München, den 8. März 1999

Deutsches Patent- und Markenamt

Der Präsident

Im Auftrag

Wehner

Aktenzeichen: 198 03 845.3

DE 97/18
BWB - 21. Januar 1998

Verfahren und Einrichtung zur komprimierten Übertragung und/oder Speicherung von einer durch digitale Daten repräsentierten Nachricht

Zusammenfassung

Es wird ein Verfahren und Einrichtungen zu seinem Vollzug vorgeschlagen, die es ermöglichen, Redundanz in digital dargestellten Nachrichten zu eliminieren.

Die Datenreduktion erfolgt dabei durch eine syntax-orientierte Codierung der Nachricht.

Verfahren und Einrichtung zur komprimierten Übertragung und/oder Speicherung von einer durch digitale Daten repräsentierten Nachricht

1 Einführung in den Themenkreis

Wichtige technische Anwendungen der Quellencodierung liegen auf dem Gebiet der Übertragung und Speicherung von Nachrichten. Aus dem Wunsch einer geringen Datenrate bei der Übertragung von Nachrichten, bzw. eines geringen Platzbedarfs bei der Speicherung von Nachrichten, erwächst die Forderung nach einer möglichst kurzen, d.h. redundanzarmen und im Idealfall redundanzfreien Codierung der Nachricht.

2 Stand der Technik

2.1 Formen von Redundanz

Redundanz tritt in Quellensymbolströmen in zweierlei grundlegenden Formen in Erscheinung: Als unterschiedliche Auftrittshäufigkeit der Elemente des Quellensymbolvorrats, und in Form von statistischen Abhängigkeiten zwischen zeitlich auseinanderliegenden Quellensymbolen, die ihre Ursache meist in quellenspezifischen *Bildungsgesetzen* für die Konstruktion von Nachrichten als Folge von Quellensymbolen haben.

3 Nachteile bekannter Verfahren

Zur Kompaktierung redundanzbehafteter Quellen, deren abgegeben Quellensymbole nicht gleichverteilt, aber statistisch unabhängig sind, ist ein Verfahren nach Huffman [1] bekannt. Speziell für diesen Quellentyp geeignet, ist es mit dem Verfahren nur unter der günstigsten Voraussetzung möglich, daß die statistischen Eigenschaften der Quelle bekannt sind oder zuverlässig geschätzt werden können, eine maximale Kompaktierung zu erreichen.

Während die Schätzung der für Huffman-Codierung notwendigen Auftrittswahrscheinlichkeiten einzelner Quellensymbole auch bei mäßig langen Nachrichten ausreichend genau möglich ist, erfordert die Schätzung statistischer Abhängigkeiten zwischen zeitlich auseinanderliegenden Quellensymbolen in der Regel eine umfassende Beobachtung der Quelle, die technische Machbarkeit (Speicherplatz) und verfügbaren Beobachtungszeitraum bei weitem übersteigen. Deswegen treffen das bekannte oder ähnliche Verfahren stets Annahmen über das zugrundeliegende Bildungsgesetz oder wesentliche Charakteristika desselben, und sind dann natürlich auch nur für Quellen mit diesem Bildungsgesetz geeignet. Diese Einschränkung gilt auch für das seit Ende der 70er Jahre bekannte Verfahren nach

Lempel-Ziv (LZ) Datenkompaktierungsverfahren [2] Dieses Verfahren ist zur Kompaktierung beliebiger linearer Symbolfolgen geeignet, d.h. es ist anwendbar ohne Kenntnis der statistischen Eigenschaften der Nachrichtenquelle. Hauptanwendungsgebiete sind heute die Kompaktierung von Bilddaten (GIF-Dateiformat) und beliebiger Dateien (Archivierungsprogramme gnuzip, pkzip). Im Gegensatz zur Huffman-Codierung nutzt dieses Verfahren, zwischen zeitlich aufeinanderfolgenden Symbolen bestehende, statistische Abhängigkeiten durch Berücksichtigung wiederholt auftretender Teilsymbolfolgen (Strings) aus. Damit ist LZ-Kompaktierung nur für solche Quellen geeignet, deren Bildungsgesetz zum häufigen Auftreten bestimmter Strings führt. Die Redundanz die durch das Befolgen grammatikalischer Regeln bei der Konstruktion einer Nachricht in diese eingebracht wird, ist den bekannten Kompaktierungsverfahren aufgrund der rekursiven Struktur grammatikalischer Regeln nicht zugänglich, da sie lediglich Symbolhäufigkeiten oder sich wiederholende Zeichenketten zur Kompaktierung heranziehen.

4 Definition der Erfindungsaufgabe

Aufgabe der Erfindung ist es, ein Verfahren und Einrichtungen zum Vollzug des Verfahrens zu schaffen, welche sich auf die Fähigkeiten erstrecken, aus einer als Zeichenstrom vorliegenden Nachricht, die gemäß grammatikalischen Regeln gebildet wurde, durch Codieren jene Redundanz zu entfernen, die durch die Einschränkung aller möglichen Zeichenfolgen durch die Grammtik entsteht.

Diese Aufgabe wird durch ein Verfahren mit dem in Anspruch 1 angegebenen Merkmalen und zwei Vorrichtungen, mit den in den Ansprüchen 8 und 9 genannten Merkmalen bewältigt.

5 Vorteile der erfinderischen Lösung

Mit der erfindungsgemäßen Lösung, eine Datenreduktion in Abhängigkeit zur verwendeten Programmiersprachensyntax vornehmen zu können, ist es nunmehr in besonders vorteilhafter Weise möglich, Redundanz in Nachrichten, die sich während ihrer Konstruktion, wegen der Notwendigkeit bestimmte grammatikalische Regeln beachten zu müssen, asymptotisch nahezu vollständig zu entfernen.

Die Erfindung hat dies durch analytische Berechnung der Kapazität ihres syntax- orientiert codierten Datenstroms am Beispiel einer Sprache mit für Programmiersprachen typischen Charakeristika (mathematische Ausdrücke, if-then-else-Konstrukt, etc.) bewiesen. Dabei zeigte sich, daß ein nicht geringer Anteil der Redundanz, der im Aufgabenbereich herkömmlicher Verfahren (Lempel-Ziv, Huffman) liegt, durch diese bisher nicht beseitigt wird.

Das neue Verfahren der syntax-orientierten Codierung ist dagegen in der Lage, die Datenmenge z.B. bezogen auf ein Komprimierungsergbnis nach dem Lempel-Ziv-Verfahren, noch einmal um nahezu die Hälfte zu mindern.

Damit ist das erfindungsgemäße Verahren und zu ihrem Vollzug definierten Einrichtungen für alle Anwendungen prädestiniert, welche die Übertragung oder Speicherung von syntaktisch strukturierten Nachrichten beinhalten. Beispiele sind die Übertragung von Java-Applets im Internet oder Intranet, die Übertragung oder Speicherung von Postscript-Dateien, die Übertragung und Speicherung von MPEG-4-codierten Videodaten, oder die Verwendung höherer Protokolle bei der Kommunikation. Gerade für die Übertragung von Programmen, wie z. B. Java-Applets, ist SoC in besonderem Maße geeignet, da der Parser sowohl Bestandteil eines Compilers als auch eines SoC-Coders ist, und somit SoC organisch in den Datenfluß von der Programmerstellung bis zum Programmablauf integriert werden kann.

6 Wirkungsweise der Erfindung

6.1 Syntaktisch strukturierte Quelle

Die Erfindung geht von einer formalen Sprache aus, welche durch eine Menge von Zeichenketten dargestellt wird. Die Zusammenstellung der Nachricht erfolgt dabei nach einer der Sprache zugeordneten Grammatik. Eine Grammatik ist sowohl das mathematische System zur Definition einer formalen Sprache, als auch ein Satz von Regeln zur Feststellung der syntaktischen Gültigkeit eines konkreten Satzes. Im folgenden werden nur kontextfreie Grammatiken betrachtet, da höhere Programmiersprachen (C, C++, Java, etc.) fast ausschließlich durch kontextfreie Grammatiken beschrieben werden. folgendermaßen definiert:

Definition 1 *Eine kontextfreie Grammatik ist ein 4-Tupel* $G = (N, T, P, S)$, *wobei*

N *die Menge der Nonterminalsymbole,*

T *die Menge der Terminalsymbole, und*

P *die Relation* $N \times (N \cup T)^*$ *ist.*

S *ist ein besonderes Nonterminalsymbol, auch Startsymbol genannt.*

Definition 2 *Ein Element* $(A, \beta) \in P$ *heißt* Produktion *und wird abkürzend* $A \rightarrow \beta$ *geschrieben.*

Eine Produktion (oder Ableitung) ist somit eine Ersetzungsregel für ein Nonterminalsymbol A, wobei dieses Nonterminalsymbol durch einen String (Kette) β aus (Nonterminal- und) Terminalsymbolen ersetzt wird.

Existieren mehrere Produktionen $(A,\beta_1),(A,\beta_2),\ldots,(A,\beta_n) \in P$, so schreiben wir hierfür:

$$\begin{array}{rcl} A & \rightarrow & \beta_1 \\ & | & \beta_2 \\ & \vdots & \\ & | & \beta_n \end{array}$$

Definition 3 *Die Menge $P_{A_0} \subseteq P$ aller möglichen Ableitungen für ein Nonterminalsymbol A_0*

$$P_{A_0} = \{(A,\beta) \in P : A = A_0\}$$

heißt auch die Menge der Alternativen *für A_0. Die Entscheidung für die Anwendung einer konkreten Produktion $(A_0,\beta_k) \in P_{A_0}, (k = 1,2,\cdots,n)$ nennen wir* Auswahl.

Beispiel 1 *Alle arithmetischen Ausdrücke in der Variablen a mit den Operationen + und $*$ sowie beliebig verschachtelten Klammern (,) sind eine formale Sprache, die durch die Grammatik*

$$G = (\{E,T,F\},\{a,+,*,(,)\},P,E)$$

erzeugt wird. P besteht aus den folgenden Produktionen:

$$\begin{array}{rcl} E & \rightarrow & E+T \\ & | & T \\ T & \rightarrow & T*F \\ & | & F \\ F & \rightarrow & (E) \\ & | & a \end{array}$$

$a*a+(a*a+a)*a$ *ist beispielsweise ein gültiger Satz.*

Bei der Erzeugung eines Satzes werden Strings aus Nonterminal- und ggf. Terminalsymbolen durch Anwendung passender Produktionen auf die Nonterminalsymbole weiter abgeleitet, bis letztendlich der String nur noch aus Terminalsymbolen besteht. Im o.g. Beispiel besteht der gültige Satz nur noch aus den Terminalsymbolen $a,+,*,(,)$.

6.2 Prinzip der Syntax-orientierten Codierung (SoC)

Information fließt in eine erzeugte Nachricht immer dann ein, wenn bei ihrer Konstruktion *Entscheidungen* getroffen werden. Entscheidungen sind dann zu treffen, wenn von mehreren möglichen Quellensymbolen eines auszuwählen ist, oder - bei einer syntaktisch strukturierten Quelle - von mehreren anwendbaren Produktionen $A \rightarrow \beta_1, A \rightarrow \beta_2, \ldots$ genau eine Produktion Anwendung findet (Auswahl, vgl. Definition 3).

SoC codiert anstelle des Strings aus Terminalsymbolen die Auswahlen, die der Reihe nach bei der Konstruktion dieses Strings (der Nachricht) vorgenommen werden.

Definition 4 $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ *heißt SoC-Alphabet mit*

$$n = \max_A \{|P_A|\}$$

Alle Elemente einer jeden Menge von Alternativen P_α werden mit SoC-Symbolen $\sigma_i \in \Sigma$ (beliebig) durchnumeriert:

Definition 5 $z_A : P_A \rightarrow \Sigma$ *ist eine (beliebige) injektive Abbildung.*

Die konkrete Definition von z_A kann i.a. abhängig von P_A sein.

Zur Veranschaulichung der Erfindung sind folgend Ausführungsbeispiele näher beschrieben und in den Figuren 1 bis 6 dargestellt. Es zeigen

- Figur 1 ein Blockschaltbild eines SoC-Übertragungssystems
- Figur 2 Alternativen der Grammatik G und mit SoC-Symbolen attributierte Auswahlen
- Figur 3 einen Parse-Baum des Satzes $a * a + (a * a + a) * a$ mit Entstehung der linearen SoC-Symbolfolge 121222121121222222
- Figur 1 ein Flußdiagramm der Stack-Maschine (Kellerautomat)
- Figur 5 ein besonderes Beispiel für ein SoC-Übertragungssystem mit empfängerseitiger Quelltextausgabe
- Figur 5 ein Beispiel für ein SoC-Übertragungssystem mit lauffähigem Java-Applet bzw. -Applikation als Ausgabe

Die Funktion eines Übertragungssystems mit SoC ist in Figur 1 veranschaulicht. Grundsätzlich sind Übertragung und Speicherung im Hinblick auf die Wirkungsweise der Codierung gleichzusetzen.

Die Quelle liefert einen syntaktisch strukturierten Symbolstrom (Quellprogramm). Elementarer Bestandteil des SoC-Coders ist der Parser, dessen Aufgabe es ist, aus der zunächst linearen Folge von Quellensymbolen die grammatikalische Struktur des Satzes (des Quellenprogramms) wiederzugewinnen. Diese grammatikalische Struktur wird i.a. durch einen Parse-Baum dargestellt, der seinerseits als Eingabe für den Coder dient. Der Coder erzeugt durch Codierung des Parse-Baumes eine lineare Folge von SoC-Symbolen, wobei das jeweils aktuelle SoC-Symbol in Abhängigkeit der aktuellen Auswahl ausgegeben und übertragen wird. Der SoC-Decoder baut aus der linearen Folge von SoC-Symbolen

zunächst den Parse-Baum wieder auf, um dann durch dessen Traversieren die von der Quelle abgegebene Folge von Terminalsymbolen zu rekonstruieren.

6.3 Codierung

Die elementare Aufgabe des Parsers besteht darin, die bei der Konstruktion des Satzes von der Quelle getroffenen *Auswahlen* zu rekonstruieren. Ein Parse-Schritt bedeutet also die Feststellung der konkreten Produktion $(A, \beta) \in P_A$, die auch die Quelle zur Konstruktion der Nachricht an dieser Stelle ausgewählt hatte. Der Coder ordnet entsprechend Definition 5 der solchermaßen festgestellten Auswahl das zugehörige SoC-Symbol $\sigma_k = z_A((A, \beta))$ zu und attributiert den entsprechenden Knoten im Parse-Baum mit σ_k.

Durch Traversieren des Parse-Baumes wird schließlich die lineare Folge von SoC-Symbolen erzeugt und ausgegeben. In den Beispielen dieses Beitrags wird hierfür stets ein Depth-first-Algorithmus verwendet. Der Prozeß der Codierung soll anhand eines beispielhaften Parse-Baumes und der zugrundeliegenden Grammatik aus Beispiel 1 näher erläutert werden.

Beispiel 2 *Gegeben ist die kontextfreie Grammatik aus Beispiel 1. Alle Alternativen, d.h. alle Ableitungsmöglichkeiten für jedes Nonterminalsymbol sind in der Figur 2 graphisch dargestellt. Da die mächtigste Menge von Alternativen genau zwei Elemente hat, genügt das SoC-Alphabet* $\Sigma = \{1, 2\}$. *Jede Auswahl aller Mengen von Alternativen wird mit einem SoC-Symbol attributiert. Beispielsweise wird die Auswahl der konkreten Produktion* $T \rightarrow T * F$ *mit dem SoC-Symbol* 1 *attributiert.*

Ein gültiger Satz der Grammatik ist dann also z. B.: $a * a + (a * a + a) * a$ *Den Parse-Baum für diesen Satz und die Entstehung der SoC-Symbolfolge durch Depth-first-Traversieren zeigt Figur 3.*

6.4 SoC mit arithmetischer Codierung

Das vorstehend erläuterte Verfahren ist nur dann effizient, wenn

1. $|P_A| = const\ \forall A$, und
2. $P((A, \beta_i)|A) = const\ \forall A, \beta_i$,

wobei $P((A, \beta_i)|\alpha)$ die bedingte Wahrscheinlichkeit der β_i erzeugenden Auswahl der Menge der Alternativen P_A bezeichnet.

Diesem Mangel kann durch arithmetische Codierung [5] des SoC-Alphabets Σ gemäß Anspruch 6 und 7 abgeholfen werden. Die für die arithmetische Codierung verwnedeten Wahrschienlichkeitsverteilungen der SoC-Symbole sollten in einer Adaptionseinrichtung nach Anspruch 10 adaptiert werden, wobei die Adaption getrennt für jede Menge von

Alternativen durchgeführt werden muß. Anstelle der linearen Sequenz von SoC-Symbolen wird der vom arithmetischen Coder erzeugte Bitstrom übertragen.

6.5 Decodierung

Die Decodierung ist der inverse Prozeß zur Codierung. Der Decoder formatiert die SoC-Codes zurück zum ursprünglichen Satz (Strings von Terminalsymbolen). Er arbeitet nach dem in den Ansprüchen 1, 3 bzw. 4, 5 beschriebenen Verfahren und wird durch eine Stack-Maschine [4] gemäß den Ansprüchen 8 bzw. 9 realisiert.

Ein Flußdiagramm für ein Verfahren nach Anspruch 22 ist in Figur 4 angegeben.

Zur Erläuterung der Stack-Maschine werden folgende Bezeichungen eingeführt:

β_k String von Terminal- und Nonterminalsymbolen

σ SoC-Symbol

V Terminal- oder Nonterminalsymbol

S Startsymbol (Nonterminalsymbol)

Die Decodierung läuft damit in folgenden Schritten ab (die Nummern korrespondieren mit den Bezugszeichen in Figur 4):

1. Die Entwicklung eines gültigen Satzes der Grammatik (eines Programms) beginnt mit dem Startsymbol S. Das Startsymbol S wird auf den Stack gelegt.
2. Das oberste Symbol V wird vom Stack gelesen.
3. Prüfen, ob V ein Nonterminalsymbol ist.
4. Vom linearen Eingangs-SoC-Symbolstrom wird das nächste SoC-Symbol σ gelesen.
5. Falls V ein Nonterminalsymbol ist, wird die Ableitung mit V weiterentwickelt. Durch σ wird eine Auswahl $(V, \beta_k) = z^{-1}(\sigma)$ der Menge der Alternativen von $V \rightarrow \beta_1|\beta_2|\cdots|\beta_n$ für V bestimmt. Entsprechend der konkreten Auswahl wird V durch den String β_k ersetzt.
6. Der String β_k von Symbolen (Terminal- und/oder Nonterminalsymbolen) wird auf den Stack gelegt.
7. Falls V ein Terminalsymbol ist, wird es ausgegeben.
8. Prüfen, ob der Stack leer ist.
9. Die Stack-Maschine terminiert, wenn der Stack leer ist.

Die sukzessive ausgegebenen Strings von Terminalsymbolen ergeben miteinander verkettet schließlich den ursprünglich codierten Satz der Quelle.

7 Ausführungsbeispiel

Die beiden in Figur 5 und 6 dargestellten Ausführungsbeispiele unterscheiden sich lediglich durch die Datenausgabe am Empfänger (Ausgabe von Quelltext entsprechend dem Verfahren nach Anspruch 3, bzw. direkte Ausgabe eines lauffähigen Java-Applets bzw. einer lauffähigen Java-Applikation entsprechend dem Verfahren nach Anspruch 4).

1. Java-Quelle

   Die Java-Quelle entspricht einem auf der Java-Grammatik beruhenden ASCII-Text, welcher aus Java-Programmtext sowie Kommentaren, Leerzeichen, usw. besteht.

2. SoC-Encoder

   Der Soc-Encoder setzt sich zusammen aus einem Compiler-Frontend ((3),(4) und (6)), einer Parse-Baum-Verarbeitung (5), einer Adaptiervorrichtung (8) und Modulen zur Kompression und Verkettung der erzeugten Daten.

3. Scanner

   Der Sanner liest den Quelltext, entfernt die darin für das Programm unwichtigen Textstellen, wie Dokumentationen, Kommentare, Leerzeichen, usw. und übergibt dem Parser (4) (Terminal-) Symbole.

4. Parser

   Der Parser interpretiert die (Terminal-) Symbole des Scanners (3), prüft ob die Symbolfolgen mit den grammatikalischen Regeln für Java übereinstimmen und erstellt den für dieses Programm spezifischen Parse-Baum. Zusätzlich wird die Symboltabelle (6) aufgebaut.

5. Parse-Baum-Verarbeitung

   Bei der Initialisierung der Parse-Baum-Verarbeitung werden alle Knoten des Parse-Baumes mit für jeden Knotentyp festgelegten SoC-Symbolen attributiert. Die darauf folgende Traversierung aller Knoten des Parse-Baumes führt bei jedem Durchschreiten eines Knotens zur Ausgabe eines SoC-Symbols. Diese SoC-Symbole werden sowohl der Adaptiervorrichtung (8) als auch dem arithmetischen Coder (10) übergeben. Zusätzlich übergibt die Parse-Baum-Verarbeitung den aktuellen Knotentyp an die Adaptiervorrichtung (8).

6. Symboltabellenspeicher

   Der Symboltabellenspeicher beinhaltet die vom Parser (4) extrahierten Bezeichner des Java-Programmtextes.

7. Zip-Kompressionseinrichtung

   Der Inhalt der Symboltabelle (6) wird durch das Verfahren nach Lempel-Zip-Welch [2] komprimiert.

8. Sendeseitige Adaptiervorrichtung

   Die Adaptiervorrichtung trägt bei der Initialisierung festgelegte Anfangswahrscheinlichkeitsverteilungen für jeden Knotentyp in der Wahrscheinlichkeitsverteilungstabelle (9) ein. Bei der laufenden Verarbeitung der von der Parse-Baum-Verabeitung (5 übernommenen SoC-Symbole und anhand der aktuellen Knotentypen wird bei jedem Schritt die Wahrscheinlichkeitsverteilung aller SoC-Symbole des aktuellen Knotentyps adaptiert und in der Wahrscheinlichkeitsverteilungstabelle (9) eingetragen. Die Adaptiervorrichtung stellt gleichzeitig dem arithmetischen Coder 10 die zum jeweils aktuellen SoC-Symbol gehörende Wahrscheinlichkeitsverteilung zur Verfügung.

9. Tabelle der Wahrscheinlichkeitsverteilungen der SoC-Symbole

   Für jeden Knotentyp enthält die Tabelle eine eigene Wahrscheinlichkeitsverteilung der SoC-Symbole des jeweiligen Knotentyps.

10. Arithmetischer Coder

    Der arithmetische Coder [5] empfängt von der Parse-Baum-Verarbeitung (5) das nächste zu codierende SoC-Symbol und codiert es unter Berücksichtigung der von der Adaptiervorrichtung (8) zur Verfügung gestellten Wahrscheinlichkeitsverteilung. Der Output des arithmetischen Coders ist ein binärer Datenstrom.

11. Datenverkettung

    Der binäre Datenstrom des arithmetischen Coders (10) und die durch (7) komprimierte Symboltabelle (6) werden zu einem Datenpaket zusammengefaßt.

12. HTTP-Server

    Auf dem HTTP-Server wird das in der Datenverkettung (11) erstellte Datenpaket abgelegt und zum Abruf auf einem Massenspeicher zur Verfügung gestellt.

13. Intranet/Internet

14. HTTP-Client (Internet-Browser)

    Der Browser hat die Aufgabe das Datenpaket über das Intranet/Internet 13 von einem HTTP-Server (12) zu laden und den Decodiervorgang durch den SoC-Decoder (15) anzustoßen.

15. SoC-Decoder

    Der SoC-Decoder besteht im wesentlichen aus dem arithmetischen Decoder (17) und dem Kellerautomaten (22). Als Ergebnis liefert er die Sequenz der (Terminal-) Symbole die dem ursprünglichen Java-Quellcode entsprechen (vgl. 3 und 4). (1).

16. Datenextraktion

Hier werden der binäre Datenstrom des arithmetischen Coders (10) und die durch Zip (7) komprimierte Symboltabelle aus dem gemeinsamen Datenpaket extrahiert und ersterer an den arithmetischen Decoder (17), letzterer an die Zip-Dekompressionseinrichtung (18) weitergegeben.

17. Arithmetischer Decoder

Der arithmetische Decoder [5] decodiert adaptiv den binären Datenstrom von der Datenextraktion (16) unter Berücksichtigung der von der empfangsseitigen Adaptiervorrichtung (20) zur Verfügung gestellten Wahrscheinlichkeitsverteilungen zu den SoC-Symbolen und übergibt pro Decodierschritt ein SoC-Symbol an den Kellerautomaten (22).

18. Zip-Dekompressionseinrichtung

Die mit mit dem Verfahren nach Lempel-Ziv (7) komprimierte Symboltabelle des Coders (6) aus der Datenextraktion (16) wird entpackt und an den Symboltabellenspeicher des Decoders (19) eingetragen.

19. Symboltabellenspeicher des Decoders

20. Adaptiervorrichtung (empfangsseitig)

Die Initialisierung der empfangsseitigen Adaptiervorrichtung erfolgt analog zur sendeseitigen Adaptiervorrichtung (8). Während der laufenden Decodierung erhält die Adaptiervorrichtung bei jedem Decodierschritt ein SoC-Symbol, adaptiert die Wahrscheinlichkeitsverteilung der SoC-Symbole des aktuellen Knotentyps und trägt die adaptierten Wahrscheinlichkeitsverteilungen in die Wahrscheinlichkeitsverteilungstabelle (21) ein. Gleichzeitig stellt die empfangsseitige Adaptiervorrichtung die für den nächsten Decodierschritt gültige Wahrscheinlichkeitsverteilung dem arithmetischen Decoder zur Verfügung. Die Auswahl dieser Verteilung bestimmt sich durch den vom Kellerautomaten (22) festgelegten und im nächsten Decodierschritt zu erwartenden Knotentyp.

21. Tabelle der Wahrscheinlichkeitsverteilungen der SoC-Symbole

Für jeden Knotentyp enthält die Tabelle eine eigene Wahrscheinlichkeitsverteilung der SoC-Symbole dieses Knotentyps.

22. Kellerautomat

Die Initialisierung des Kellerautomaten, die Bearbeitung des Stapelspeichers, sowie die Ausgabe von Terminalsymbolen ist in Figur 4 als Flußdiagramm dargestellt. Zusätzlich übergibt der Kellerautomat Daten an die Adaptiervorrichtung(20):

(a) den jeweils aktuellen Knotentyp zur Adaption der Wahrscheinlichkeitsverteilungen der SoC-Symbole dieses Knotentyps

(b) den nächsten Knotentyp, dessen Wahrscheinlichkeitsverteilung der SoC-Symbole die Adaptiervorrichtung (20) im nächsten Decodierschritt dem arithmetischen Decoder (17) zur Verfügung stellen muß.

23. Stapelspeicher

Stapelspeicher (Stack) für den Kellerautomaten (22).

24. Java-Sinke

Wiedergewonnener Java-Programmtext ohne Kommentare, überflüssige Leerzeichen, usw. .

25. Codeerzeuger

Der Codeerzeuger enspricht einem Compiler-Backend und generiert Bytecode für die JVM (26) in Abhängigkeit der Sequenz der (Terminal-) Symbole, die der Kellerautomat (22) liefert.

26. JVM

Die virtuelle Maschine für Java-Programme (Java Virtual Machine).

Literatur

[1] HUFFMAN, D. A.: *A Method for the Construction of Minimum-Redundancy Codes.* Proc. IRE, 40:1098–1101, 1952.

[2] ZIV, JACOB und ABRAHAM LEMPEL: *A Universal Algorithm for Sequential Data Compression.* IEEE Trans. Inform. Theory, IT-23(3):337–343, Mai 1977.

[3] AHO, ALFRED V. und JEFFREY D. ULLMAN: *The Theory of Parsing, Translation and Compiling.* Prentice Hall Inc., New Jersey, 1972.

[4] AHO, ALFRED V., RAVI SETHI und JEFFREY D. ULLMAN: *Compilerbau.* Addison-Wesley Publishing Company, Bonn, 4. Auflage, 1988.

[5] WITTEN, IAN H., RADFORD M. NEAL und JOHN G. CLEARY: *Arithmetic Coding for Data Compression.* Commun. ACM, 30(6):520–540, Juni 1987.

Verfahren und Einrichtung zur komprimierten Übertragung und/oder Speicherung von einer durch digitale Daten repräsentierten Nachricht

Bezugszeichenliste

Bezugszeichen zur Figur 4

β_k String von Terminal- und Nonterminalsymbolen
σ SoC-Symbol
V Terminal- oder Nonterminalsymbol
S Startsymbol (Nonterminalsymbol)

1. Die Entwicklung eines gültigen Satzes der Grammatik (eines Programms) beginnt mit dem Startsymbol S. Das Startsymbol S wird auf den Stack gelegt.
2. Das oberste Symbol V wird vom Stack gelesen.
3. Prüfen, ob V ein Nonterminalsymbol ist.
4. Vom linearen Eingangs-SoC-Symbolstrom wird das nächste SoC-Symbol σ gelesen.
5. Falls V ein Nonterminalsymbol ist, wird die Ableitung mit V weiterentwickelt. Durch σ wird eine Auswahl $(V, \beta_k) = z^{-1}(\sigma)$ der Menge der Alternativen von $V \rightarrow \beta_1|\beta_2|\cdots|\beta_n$ für V bestimmt. Entsprechend der konkreten Auswahl wird V durch den String β_k ersetzt.
6. Der String β_k von Symbolen (Terminal- und/oder Nonterminalsymbolen) wird auf den Stack gelegt.
7. Falls V ein Terminalsymbol ist, wird es ausgegeben.
8. Prüfen, ob der Stack leer ist.
9. Die Stack-Maschine terminiert, wenn der Stack leer ist.

Bezugszeichen zu den Figuren 5 und 6

1. Java-Quelle
2. SoC-Encoder
3. Scanner
4. Parser
5. Parse-Baum-Verarbeitung
6. Symboltabellenspeicher
7. Zip-Kompressionseinrichtung
8. Sendeseitige Adaptiervorrichtung
9. Tabelle der Wahrscheinlichkeitsverteilungen der SoC-Symbole

10. Arithmetischer Coder
11. Datenverkettung
12. HTTP-Server
13. Intranet/Internet
14. HTTP-Client (Internet-Browser)
15. SoC-Decoder
16. Datenextraktion
17. Arithmetischer Decoder
18. Zip-Dekompressionseinrichtung
19. Symboltabellenspeicher des Decoders
20. Adaptiervorrichtung (empfangsseitig)
21. Tabelle der Wahrscheinlichkeitsverteilungen der SoC-Symbole
22. Kellerautomat
23. Stapelspeicher
24. Java-Sinke
25. Codeerzeuger
26. JVM

Patentansprüche

1. Verfahren zur komprimierten Übertragung und/oder Speicherung von einer durch digitale Daten repräsentierten Nachricht, deren Struktur durch eine Grammatik definiert wird,

   dadurch gekennzeichnet, daß

   (a) vor der Speicherung und/oder Übertragung die Nachricht in eine lineare Sequenz von Symbolen, welche die sukzessive Anwendung grammatikalischer Regeln zur Bildung der Nachricht definiert, überführt wird, und

   (b) nach der Speicherung und/oder Übertragung zur Decodierung für jedes so syntax orientiert codierte (SoC) Symbol die dem jeweiligen SoC-Symbol entsprechende grammatikalische Regel ausgeführt wird und durch diese Regel bestimmte Ausgangsdaten erzeugt werden.

2. Verfahren nach Anspruch 1,

   dadurch gekennzeichnet, daß zur Erzeugung der SoC-Symbole

   (a) durch Parsen der Nachricht ein den aufeinanderfolgenden Anwendungen der grammatikalischen Regeln entsprechender Parse-Baum erzeugt wird,

   (b) jeder Knoten dieses Parse-Baumes mit einem SoC-Symbol attributiert wird, welches unter allen an dieser Stelle durch die Grammatik erlaubten Regeln die tatsächlich zur Erzeugung der ursprünglichen Nachricht angewendete Regel eindeutig identifiziert, und

   (c) die SoC-Symbole gemäß einer festgelegten Reihenfolge der Traversierung aller Knoten des Parse-Baumes zu einer linearen Sequenz von SoC-Symbolen verkettet werden.

3. Verfahren nach Anspruch 1,

   dadurch gekennzeichnet, daß

   zur Decodierung

   (a) der oberste Eintrag eines Stapelspeichers gemäß der durch das eingegebene SoC-Symbol bestimmten Produktion der Grammatik substituiert wird,

   (b) noch nicht vollständig bearbeitbare Teile der grammatikalischen Regel in einem Stapelspeicher ablegt werden, und

   (c) vollständig substituierte Teile der Produktion als Teil der ursprünglichen Nachricht ausgegeben werden.

4. Verfahren nach Anspruch 1,

   dadurch gekennzeichnet, daß

   zur Decodierung

(a) der oberste Eintrag eines Stapelspeichers gemäß der durch das eingegebene SoC-Symbol bestimmten Produktion der Grammatik substituiert wird,

(b) noch nicht vollständig bearbeitbare Teile der grammatikalischen Regel in einem Stapelspeicher ablegt werden.

(c) vollständig substituierte Teile der Produktion unmittelbar zu einem Teil eines ausführbaren Programms für einen realen Prozessor oder eine virtuelle Maschine verarbeitet werden.

5. Verfahren nach Anspruch 3 oder 4

dadurch gekennzeichnet, daß ein Kellerautomat

(a) durch das Ablegen eines definierten Startsymbols (Nonterminalsymbol) auf den leeren Stapelspeicher initialisiert wird,

(b) das oberste Symbol vom Stapelspeicher liest,

(c) prüft, ob das gelesene Symbol ein Terminal- oder Nonterminalsymbol ist,

(d) falls es ein Terminalsymbol ist, das Symbol ausgibt und abhängig davon, ob weitere Symbole im Stapelspeicher vorhanden sind, mit 5b fortfährt bzw. terminiert, falls der Stapelspeicher leer ist, oder

(e) falls es ein Nonterminalsymbol ist, das nächste SoC-Symbol vom Eingabestrom liest,

(f) abhängig vom gelesenen SoC-Symbol genau eine Alternative (Kette von Terminal- und/oder Nonterminalsymbolen) aus der für das aktuell bearbeitete Nonterminalsymbol gültigen Menge von alternativ anwendbaren Ersetzungsregeln (Produktionen) auswählt und

(g) diese Kette von Terminal- und/oder Nonterminalsymbolen auf den Stapelspeicher legt und mit 5b fortfährt.

6. Verfahren nach Anspruch 2,

dadurch gekennzeichnet, daß

(a) jeder Knoten des Parse-Baumes mit einem SoC-Symbol und der Wahrscheinlichkeitsverteilung aller in diesem Knoten möglichen SoC-Symbole attributiert wird,

(b) die lineare Sequenz der SoC-Symbole in Einheit mit den zugeordneten Wahrscheinlichkeitsverteilungen einer Entropie-Codierung unterworfen wird,

(c) die Entropie-Decodierung mit den identischen Wahrscheinlichkeitsverteilungen der SoC-Symbole durchgeführt wird, wie sie bei der Entropie-Codierung angewendet wurden.

7. Verfahren nach Anspruch 6,

dadurch gekennzeichnet, daß

(a) die Wahrscheinlichkeitsverteilung der in einem Knoten anwendbaren Regeln, ausgehend von einer Anfangsverteilung, bei jedem Auftreten eines SoC-Symbols so adaptiert wird, daß die Wahrscheinlichkeit für das aufgetretene SoC-Symbol erhöht und die Wahrscheinlichkeit aller anderen Symbole entsprechend verringert wird,

(b) die jeweils aktuelle Verteilung der Auftrittswahrscheinlichkeiten den SoC-Symbolen des entsprechenden Knotentyps zugeordnet wird,

(c) die Wahrscheinlichkeitsverteilung aller SoC-Symbole im aktuellen Knoten zusammen mit dem zu codierenden SoC-Symbol das Modell für eine arithmetische Codierung bildet,

(d) bei der Decodierung das Ende der Nachricht dadurch erkannt wird, daß der Stapelspeicher leer ist und

(e) auf das bei der arithmetischen Codierung notwendige End-Of-Message (EOM) Symbol verzichtet wird.

8. Einrichtung für das Verfahren nach Anspruch 1

gekennzeichnet durch

einen Codierer, bestehend aus

(a) einem Scanner zur Umformung der Nachricht als Sequenz von lesbaren Zeichen in eine Folge von Terminalsymbolen,

(b) einem Parser zum Auffinden der grammatikalischen Regeln, durch deren sukzessive Anwendung die Folge von Terminalsymbolen ursprünglich erzeugt wurde,

(c) einem Mapper, der den vom Parser identifizierten Regeln eindeutig Syntax-orientierte Symbole zuordnet und diese in festgelegter Reihenfolge ausgibt,

und einen Decodierer, bestehend aus

(a) einem Kellerautomaten, der, entsprechend dem obersten Symbol des Stapelspeichers und ggf. dem anliegenden SoC-Symbol, das bereits feststehende Terminalsymbol ausgibt, oder die dem aktuellen Symbol zugeordnete Sequenz von Terminal- und/oder Nonterminalsymbolen auf dem Stapelspeicher ablegt und

(b) einem Lexikon, das die Terminalsymbole wieder durch Ketten aus lesbaren Zeichen ersetzt.

9. Einrichtung für das Verfahren nach Anspruch 3

gekennzeichnet durch

einen Codierer, bestehend aus

(a) einem Scanner zur Umformung eines, im Quelltext oder einer vom Quelltext durch einen Präprozessor abgeleiteten Form vorliegenden, Programms in eine Folge von Terminalsymbolen,

(b) einem Parser zum Auffinden der grammatikalischen Regeln, durch deren sukzessive Anwendung die Folge von Terminalsymbolen ursprünglich erzeugt wurde,

(c) einem Mapper, der den vom Parser identifizierten Regeln eindeutig Syntax-orientierte Symbole zuordnet und diese in festgelegter Reihenfolge ausgibt,

und einen Decodierer, bestehend aus

(a) einem Kellerautomaten, der, entsprechend dem obersten Symbol des Stapelspeichers und ggf. dem anliegenden SoC-Symbol, das bereits feststehende Terminalsymbol ausgibt, oder die dem aktuellen Symbol zugeordnete Sequenz von Terminal- und/oder Nonterminalsymbolen auf dem Stapelspeicher ablegt und

(b) einem Codegenerator, der aus der Folge von Terminalsymbolen ausführbaren Maschinencode, oder Zwischencode zur Ausführung auf einer virtuellen Maschine erzeugt.

10. Einrichtung für das Verfahren nach Anspruch 7

bestehend aus

(a) senderseitig

i. einer Tabelle, die die Wahrscheinlichkeitsverteilungen der SoC-Symbole für jeden Knotentyp beinhaltet und deren Inhalt bei der Initialisierung mit festgelegten Anfangswahrscheinlichkeitsverteilungen für jeden Knotentyp belegt wird,

ii. einer Adaptionsvorrichtung, die die Wahrscheinlichkeitsverteilung der SoC-Symbole des momentan gültigen Knotentyps anhand der bestehenden Wahrscheinlichkeitsverteilung, des zu codierenden SoC-Symbols und des aktuellen Knotentyps aktualisiert und diese neue Wahrscheinlichkeitsverteilung in die Tabelle einträgt,

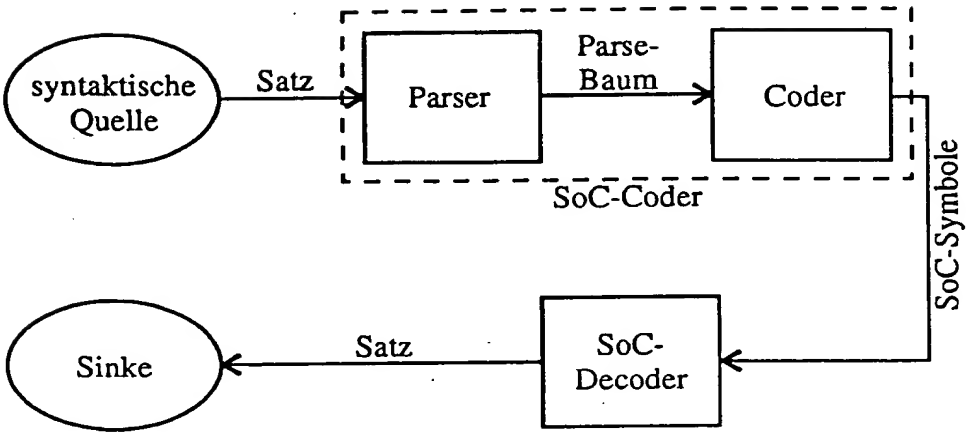
iii. und einer arithmetischen Codiervorrichtung, die das jeweils zu codierende SoC-Symbol mit der von der Adaptionsvorrichtung zur Verfügung gestellten aktuell gültigen Wahrscheinlichkeitsverteilung codiert;

(b) und empfängerseitig

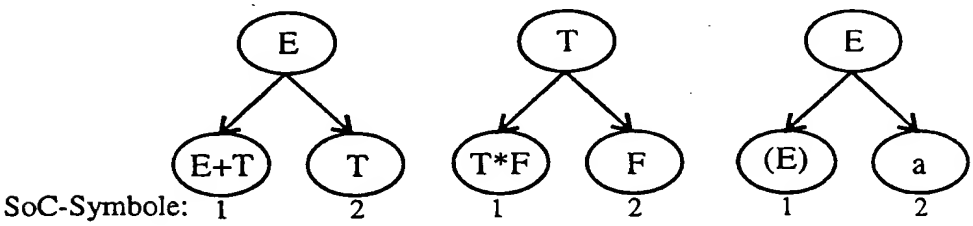
i. einer Tabelle, die die Wahrscheinlichkeitsverteilungen der SoC-Symbole für jeden Knotentyp beinhaltet und deren Inhalt bei der Initialisierung mit festgelegten Anfangswahrscheinlichkeitsverteilungen für jeden Knotentyp belegt wird,

ii. einer Adaptionsvorrichtung, die die Wahrscheinlichkeitsverteilung der SoC-Symbole des vom Kellerautomaten festgelegten, momentan gültigen Knotentyps anhand der bestehenden Wahrscheinlichkeitsverteilung, des decodierten SoC-Symbols und des aktuellen Knotentyps aktualisiert und diese neue Wahrscheinlichkeitsverteilung in die Tabelle einträgt,

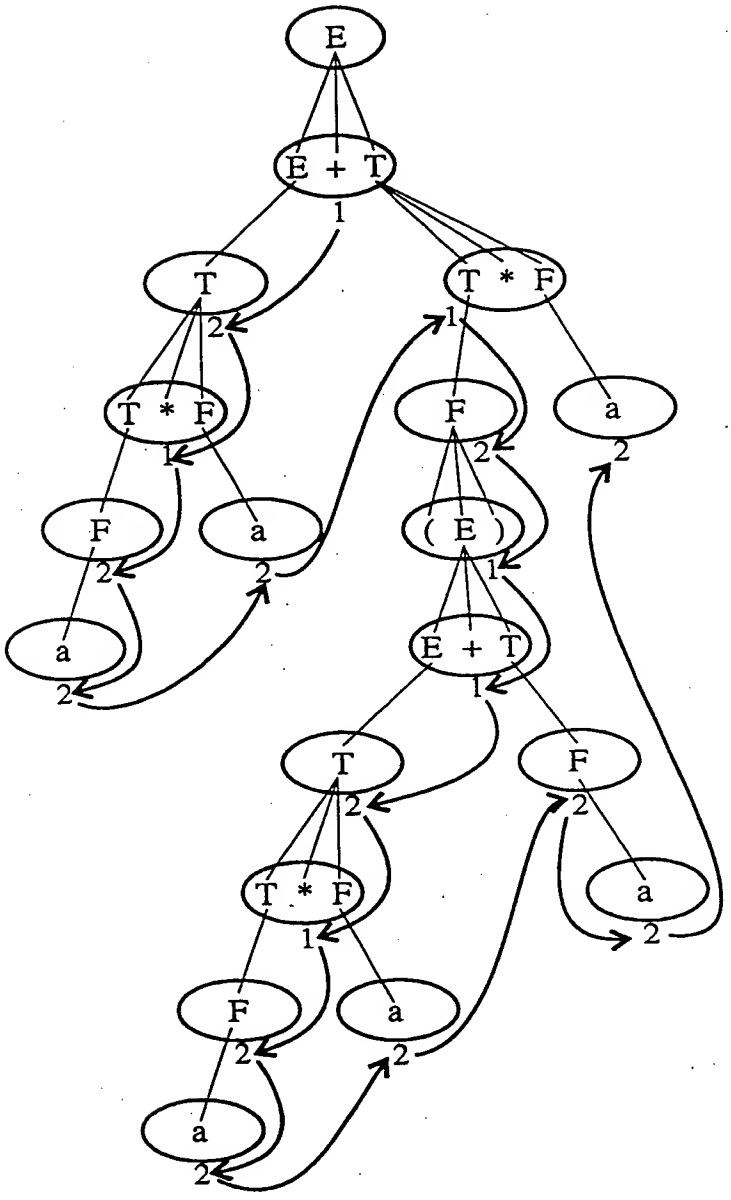
iii. einer arithmetischen Decodiervorrichtung, die anhand der von der Adaptionsvorrichtung zur Verfügung gestellten, aktuell gültigen Wahrscheinlichkeitsverteilung des aktuellen Knotentyps das nächste SoC-Symbol decodiert und zur weiteren Verarbeitung dem Kellerautomaten übergibt.



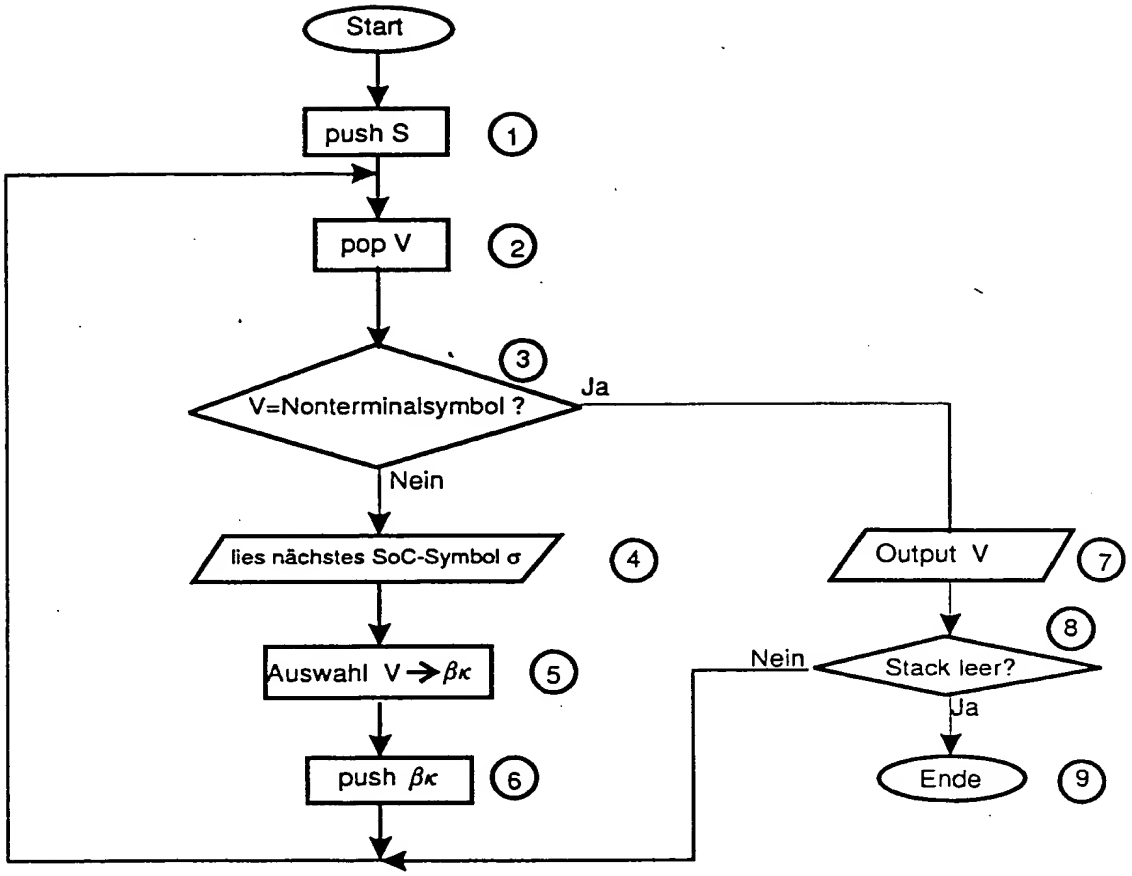
Figur 1



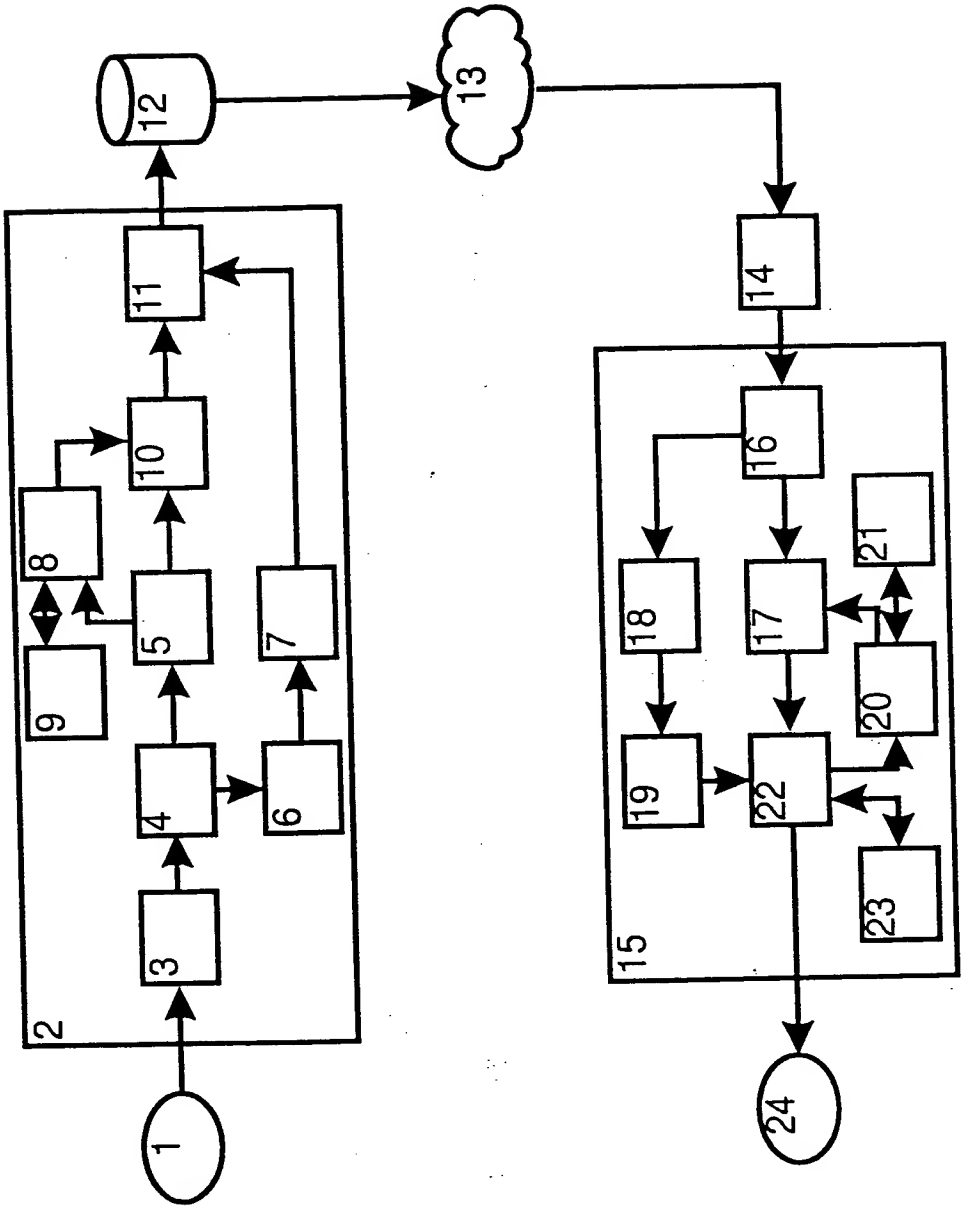
Figur 2



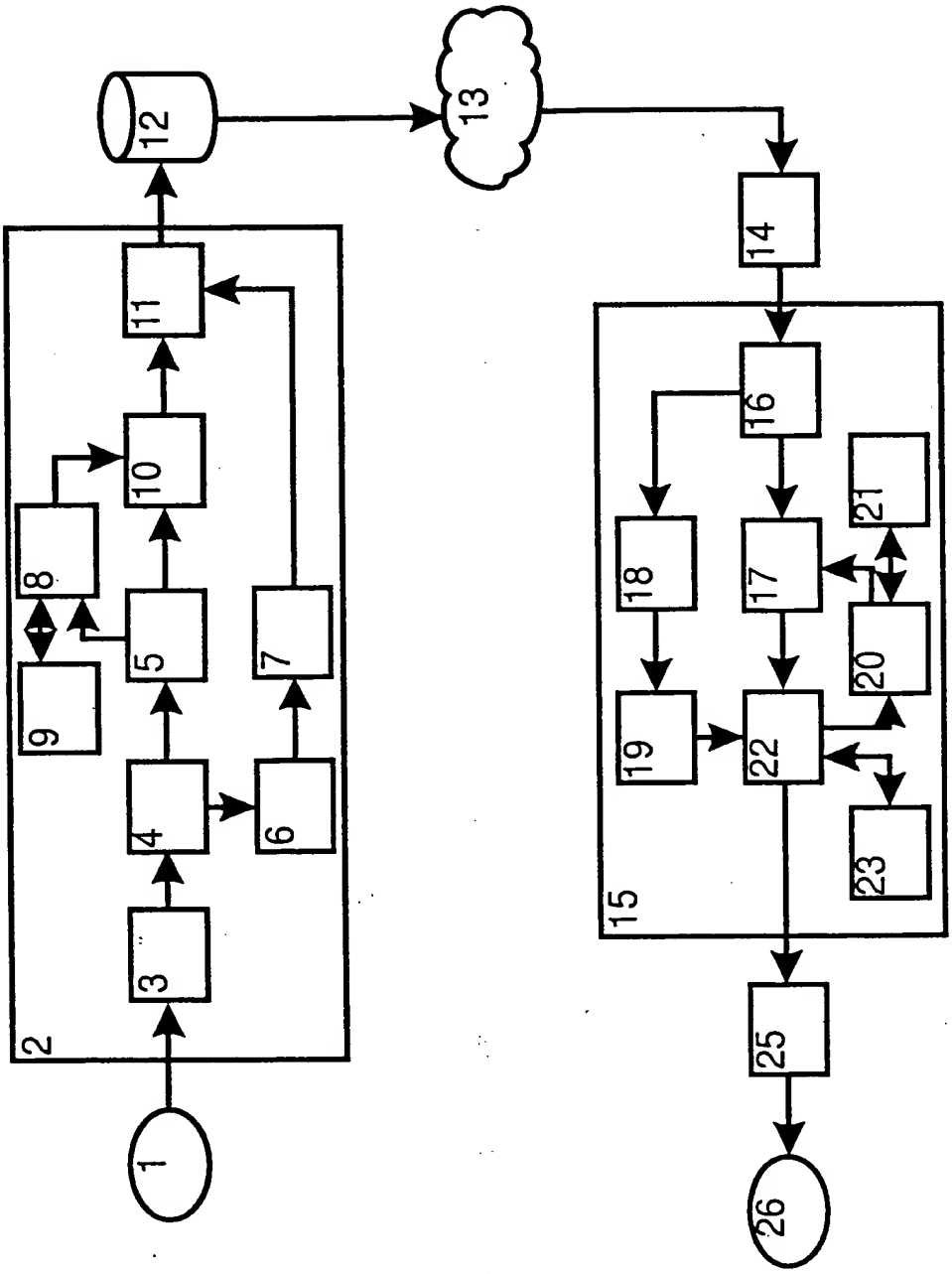
Figur 3


Start
push S
1
pop V
2
3
V=Nonterminalsymbol ?
Ja
Nein
lies nächstes SoC-Symbol σ
4
Output V
7
8
Auswahl V → βκ
5
Nein
Stack leer?
Ja
push βκ
6
Ende
9

Figur 4



Figur 5



Figur 6